

Rapid Introduction to Web Programming with Apache

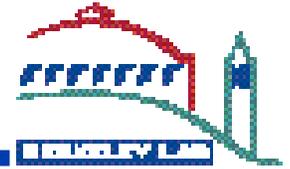
James Buszard-Welcher

jwelcher@lbl.gov

**UNIX Systems Engineer
Information Technologies & Services Division**

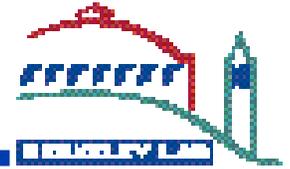
June 6th, 2003

Overview



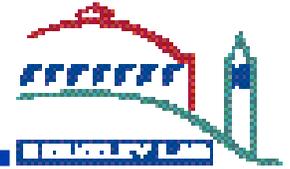
Starting from scratch, let's investigate apache, HTML, HTTP and build up to understanding how to produce dynamic web content and writing some basic web applications.

What this talk is about

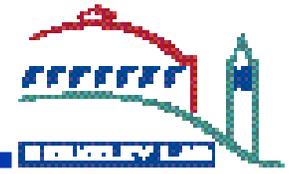


- **Compiling/Install Apache**
- **Configuring Apache**
- **Web Standards**
- **Web Server Programming**

What this talk is not about



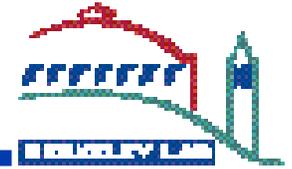
- **Web client programming (javascript, JAVA, Shockwave Flash, etc.)**
- **Any particular programming language**
- **XML/XSL/CSS/DHTML, etc.**



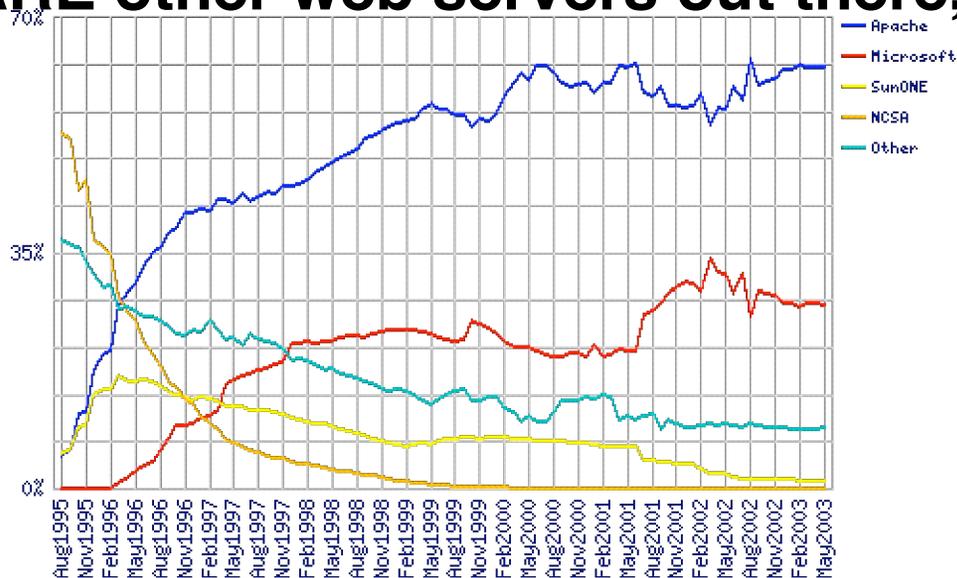


Compiling / Installing Apache

Why Apache?

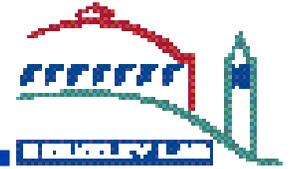


- You already know because you are here and not in the Microsoft track.
- Corporate support: Xerox, Oracle, etc.
- Prebuilt packages and/or included in most Oses
- There **ARE** other web servers out there, though...



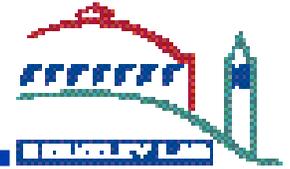
Web Server survey courtesy of Netcraft

Which Apache?



- **1.3.27 vs. 2.0.46?**
- **Apache 2 offers:**
 - **Hybrid thread/process capability**
 - **Filtered I/O**
 - **Built-in SSL**
 - **IPv6!!**
- **Apache 2 also offers:**
 - **Bleeding edge technology**
 - **Frequent bugfixes, security updates**
 - **Development version of mod_perl**

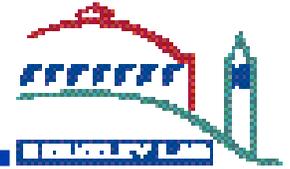
Apache Binaries



- **RedHat: RPM included with distribution**
- **Mac OS X: Includes Apache**
- **Solaris: Sunfreeware.com**
- ***BSD: ports and packages**

But you will probably want to compile it yourself...

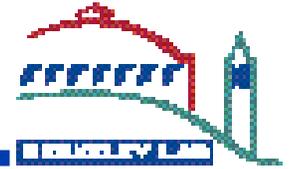
Compiling Apache



It's as easy as APACI, DSO, APXS:

- **Configure with APACI (./configure)**
- **Use Dynamic Shared Objects (DSO)**
- **Add extra modules with APXS**

Retrieve Apache and Validate



DEMO REVIEW

```
wget http://www.apache.org/dist/httpd/apache\_1.3.27.tar.gz
```

```
wget http://www.apache.org/dist/httpd/apache\_1.3.27.tar.gz.asc
```

```
cat apache_1.3.27.tar.gz | gpg --verify apache_1.3.27.tar.gz.asc -
```

```
# Either use this wget or go through web http://pgp.mit.edu
```

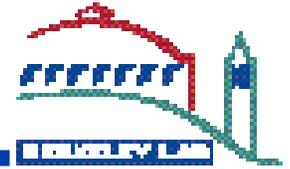
```
wget -O impkey.txt \
```

```
  'http://pgp.mit.edu:11371/pks/lookup?op=get&search=0x08C975E5'
```

```
gpg --import impkey.txt
```

```
cat apache_1.3.27.tar.gz | gpg --verify apache_1.3.27.tar.gz.asc -
```

Canonical Build Instructions



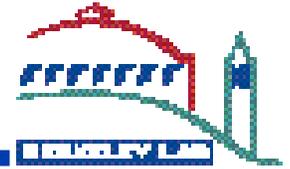
Once you unpack the tarball:

**Check out README, INSTALL, and
./configure --help**

**There are also man pages and TONS on stuff on the
web, either officially at <http://httpd.apache.org> or
via google.**

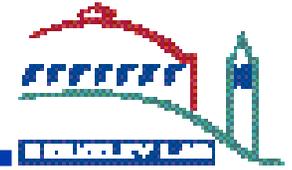
We'll skip these naturally...

Default Apache Modules



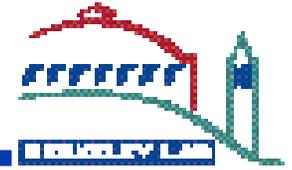
access=yes	actions=yes	alias=yes
asis=yes	auth=yes	auth_anon=no
auth_db=no	auth_dbm=no	auth_digest=no
autoindex=yes	cern_meta=no	cgi=yes
digest=no	dir=yes	env=yes
example=no	expires=no	headers=no
imap=yes	include=yes	info=no
log_agent=no	log_config=yes	log_referer=no
mime=yes	mime_magic=no	mmap_static=no
negotiation=yes	proxy=no	rewrite=no
setenvif=yes	so=no	speling=no
status=yes	unique_id=no	userdir=yes
usertrack=no	vhost_alias=no	

EZ Compile Recipe



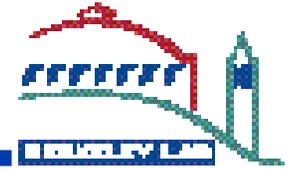
```
tar xzf apache_1.3.27.tar.gz
cd apache_1.3.27
./configure --prefix=/usr/local/apache \
            --enable-shared=max --enable-module=most
make
make install
```

Slightly Fancier Compile Recipe



```
tar xzf apache_1.3.27.tar.gz
cd apache_1.3.27
echo "./configure --prefix=/usr/local/apache \
      --enable-shared=max --enable-module=most
" > ../config_apache
. ../config_apache 2>&1 | tee config.out
time make 2>&1 | tee make.out
make install 2>&1 | tee install.out
```

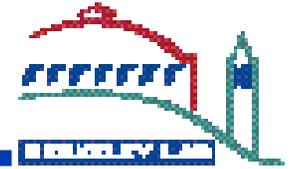
Adding a sample module: mod_macro.c



```
wget http://www.cri.enscm.fr/~coelho/mod_macro/mod_macro-1.1.2.tar.gz
tar xzf mod_macro-1.1.2
cd apache_1.3.27
./configure --prefix=/usr/local/apache \
            --enable-shared=max --enable-module=most \
            --add-module=../mod_macro-1.1.2/mod_macro.c \
            --enable-shared=macro

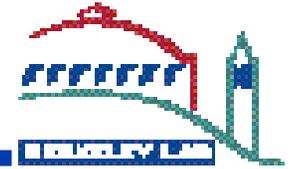
make
make install
```

Adding a module with APXS



```
tar zxf php-4.3.2.tar.bz2
cd php-4.3.2
./configure --with-apxs
make
make install
```

Important Modules

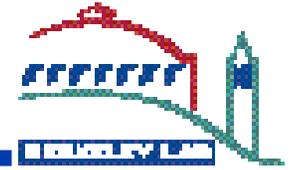


- **mod_ssl (pretty important)**
 - openssl
 - mm
- **mod_perl (highly useful)**

Others

- **PHP**
- **mod_macro (I like it)**
- **mod_ldap**
- **Many many others...**

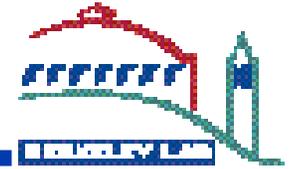
Complete Recipe p1



```
#!/bin/sh
## Get and verify apache, openssl, mm, mod_ssl, mod_perl
##
mkdir src build
cd src

wget http://www.apache.org/dist/httpd/apache_1.3.27.tar.gz
wget http://www.apache.org/dist/httpd/apache_1.3.27.tar.gz.asc
cat apache_1.3.27.tar.gz | gpg --verify apache_1.3.27.tar.gz.asc -
# Either use this wget or go through web interface
wget -O apachekey.txt \
    'http://pgp.mit.edu:11371/pks/lookup?op=get&search=0x08C975E5'
gpg --import apachekey.txt
cat apache_1.3.27.tar.gz | gpg --verify apache_1.3.27.tar.gz.asc -
wget http://www.openssl.org/source/openssl-0.9.7b.tar.gz
wget --passive-ftp ftp://ftp.ossfp.org/pkg/lib/mm/mm-1.3.0.tar.gz
wget http://www.cri.ensmp.fr/~coelho/mod_macro/mod_macro-1.1.2.tar.gz
wget http://www.modssl.org/source/mod_ssl-2.8.14-1.3.27.tar.gz
wget http://www.modssl.org/source/mod_ssl-2.8.14-1.3.27.tar.gz.asc
cat mod_ssl-2.8.14-1.3.27.tar.gz | \
    gpg --verify mod_ssl-2.8.14-1.3.27.tar.gz.asc -
wget -O modsslkey.txt \
    'http://pgp.mit.edu:11371/pks/lookup?op=get&search=0x26BB437D'
gpg --import modsslkey.txt
gpg --edit-key 0x26BB437D # trust (marginally) (sign) - OPTIONAL
cat mod_ssl-2.8.14-1.3.27.tar.gz | \
    gpg --verify mod_ssl-2.8.14-1.3.27.tar.gz.asc -
```

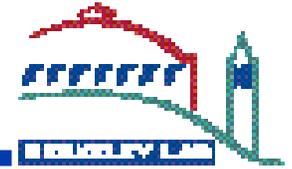
Complete Recipe p2



```
wget http://perl.apache.org/dist/mod_perl-1.27.tar.gz
wget http://perl.apache.org/dist/mod_perl-1.27.tar.gz.asc
cat mod_perl-1.27.tar.gz | gpg --verify mod_perl-1.27.tar.gz.asc -
wget -O modperlkey.txt \
    'http://pgp.mit.edu:11371/pks/lookup?op=get&search=0x6664C078'
gpg --import modperlkey.txt
cat mod_perl-1.27.tar.gz | gpg --verify mod_perl-1.27.tar.gz.asc -

## EXTRACT
##
cd ../build
tar zxf ../src/apache_1.3.27.tar.gz
tar zxf ../src/openssl-0.9.7b.tar.gz
tar zxf ../src/mm-1.3.0.tar.gz
tar zxf ../src/mod_macro-1.1.2.tar.gz
tar zxf ../src/mod_ssl-2.8.14-1.3.27.tar.gz
tar zxf ../src/mod_perl-1.27.tar.gz
```

Complete Recipe p3

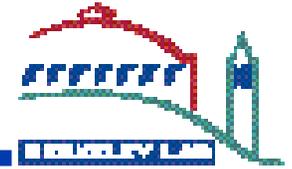


```
## BUILD
##
cd openssl-0.9.7b
./config -fPIC no-threads 2>&1 | tee config.out
make                2>&1 | tee make.out
make test          2>&1 | tee maketest.out
cd ..

cd mm-1.3.0
./configure --disable-shared 2>&1 | tee configure.out
make                2>&1 | tee make.out
cd ..

cd mod_ssl-2.8.14-1.3.27
./configure --with-apache=../apache_1.3.27 \
            --with-ssl=../openssl-0.9.7b \
            --with-mm=../mm-1.3.0 2>&1 | tee configure.out
cd ..
```

Complete Recipe p4

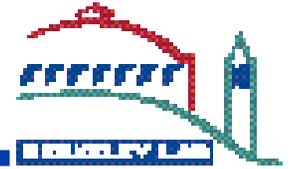


```
cat >makepl_args.mod_perl <<_END_
## This file read in when doing "perl Makefile.PL" in mod_perl dir,
## and will automatically compile apache as well. you should have
## already done a configure --with-apache=../apache... in the
## mod_ssl directory.
##
## makepl_args.mod_perl

    SSL_BASE=../openssl-0.9.7b           \\
    APACHE_SRC=../apache_1.3.27/src      \\
    APACHE_PREFIX=/usr/local/apache-1.3.27 \\
    DO_HTTPD=1                          \\
    USE_APACI=1                          \\
    EVERYTHING=1                        \\
    APACI_ARGS=--enable-module=ssl,--enable-shared=ssl,--enable-module=most,
--enable-shared=max,--enable-shared=perl,--add-module=../mod_macro-1.1.2/mod_macro.c,--enable-shared=macro
_END_
cd mod_perl-1.27
perl Makefile.PL 2>&1 | tee perlmake.out
make 2>&1 | tee make.out
make test 2>&1 | tee test.out

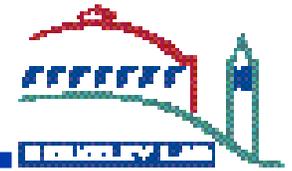
echo "as root: make install 2>&1 | tee install.out"
ln -s /usr/local/apache-1.3.27 /usr/local/apache
```

Apache Install References



<http://httpd.apache.org/docs>

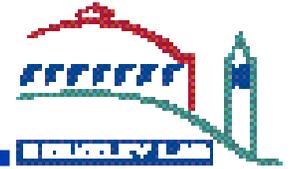




Configuring Apache

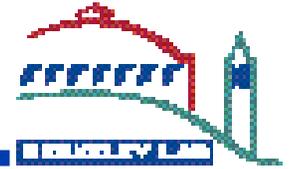
Quick and Dirty Overview

httpd.conf



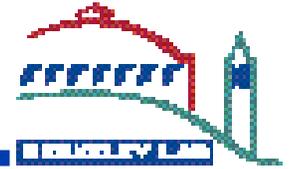
- **Very well commented. Read it.**
- **Be prepared to consult the docs for the meanings of various directives.**
- **Some directives are embedded into others.**
- **You can allow directives to be overridden by files in the content tree: .htaccess files**

Important Directives



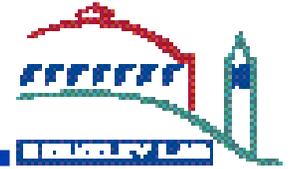
- **ServerName**
- **Port**
- **DocumentRoot**
- **Alias: map URL to directory**
- **ScriptAlias: map URL to a directory containing executables**
- **LoadModule v. AddModule**
- **Virtual Host**

LoadModule v. AddModule



- **LoadModule** is to link in DSOs
- **Compiled in modules can be inactive. The AddModule directive is used to enable them.**

Scope Directives



- **<Directory ...> ... </Directory> : Real System Path**

```
Alias /icons/ "/usr/local/apache/share/httpd/icons/"  
<Directory "/usr/local/apache/share/httpd/icons">
```

- **<Location ...> ...</Location> : Virtual URL**

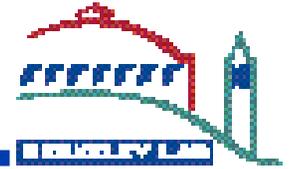
```
<Location /news>
```

- **<Files ...> ... </Files> : match file regexp**

```
<Files ~ "\.(gif|jpe?g|png)$">
```

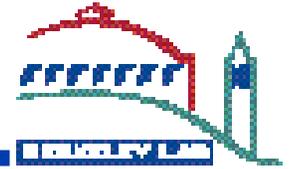
- **<VirtualHost>**

kStarting Apache



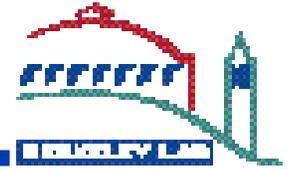
apachectl wrapper
Or just httpd

Restricting Access



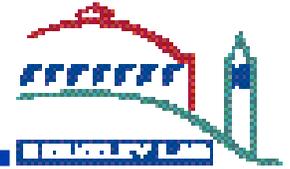
```
<Directory /web/cisdemo/htdocs/admin >  
    Order deny,allow  
    Satisfy any  
    AuthType Basic  
    AuthName "Admins Only"  
    AuthUserFile /web/cisdemo/etc/users  
    Require valid-user  
    Deny from all  
</Directory>
```

.htaccess file



- **A text file containing more directives that apply to the directory containing the file and it's sub-directories**
- **Useful for granting content owners control of their own sub-trees**
- **Can be a performance issue due to recursive nature of scope**

Logging



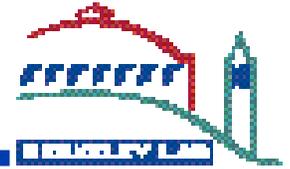
- **By default, access_log and error_log**
- **Probably want to use “combined” log format to get Referer (sic) and User-Agent data**
- **Adjustable error reporting detail**

```
ErrorLog /usr/local/apache/logs/error_log
```

```
LogLevel warn
```

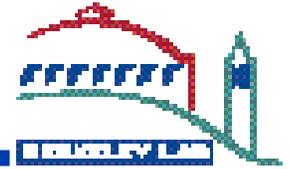
```
CustomLog /usr/local/apache/logs/access_log combined
```

Inside the Log files



- **Access**
 - Records each HTTP transfer
 - Optionally collect browser and referer data
- **Error**
 - Useful to babysit

Virtual Hosts



- **To make one daemon respond to multiple server names. Server pays attention to the name it was called.**
- **Not as useful until advent of HTTP 1.1 which added a “Host:” header, previously had to run multiple daemons bound to distinct IP addresses**
- **Name vs. IP Virtual hosts**

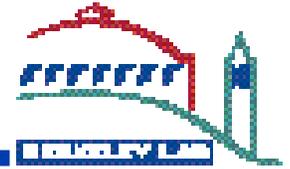




General Web Concepts

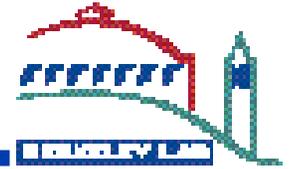
HTML and HTTP as building blocks

Know your client: HTML



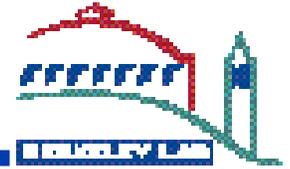
- **Who has NOT done ANY HTML from scratch?**
- **Who has NOT used the “View Source” command in their browser?**
- **Simplest approach: least common denominator
All browsers get same content, so make it easy to understand**

HTML History



- **Tim Berners-Lee at CERN circa 1991 dumbed down SGML (Standard Generalized Markup Language) to create HTML (HyperText Markup Language) for creating and linking documents across the Internet.**
- **HTTP was the very simple network protocol created to distribute it.**
- **Current is 4.01**
- **Looks like XHTML will replace, as newer browsers support XML**

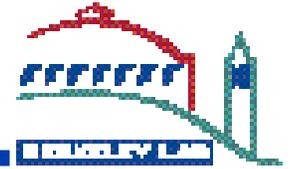
Hand Rolled HTML Follows



There are lots of applications for generating HTML with fancy WYSIWYG interfaces.

Don't Use Them! Well, at least when you are learning or teaching HTML. IMHO YMMV ;-)

Elements



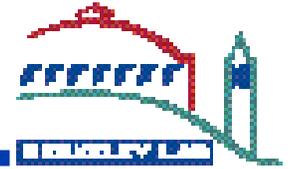
- HTML tags (ELEMENTS) describe the content
- The `<head>...</head>` contains metadata
- The `<body>...</body>` contains the content
- Some of the basic elements are:
 - paragraphs: `<p>`
 - headers: `<h1></h1>` .. `<h5></h5>`
 - ordered (``) and unordered (``) lists

Attribute Tags



- Tags can have “attributes”
- Anchor tags are what link HTML pages to one another (thus creating the Web) using hypertext reference attributes:
`Look here`
- Images included in web pages are separate documents, but are included with the image tag:
``

Entities



An ENTITY is a way of encoding a character that is special to HTML. Examples:

To say:

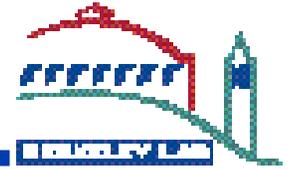
5 > 3

You must write in HTML:

5 > 3

Because “>” is “special” to HTML.

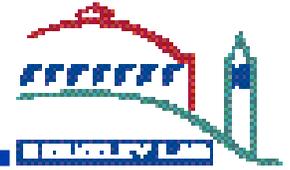
Comments



Very strange looking:

**<!-- This is a comment and can contain tags like this <P>hello
 and can cross lines, until you end the comment with something like this: -->**

HTML Hello World



`http://cisdemo.lbl.gov/standards/hello_world.html`

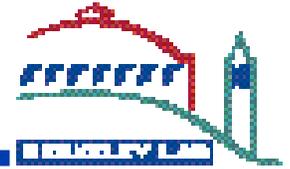
```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
</HEAD>

<BODY>
<H1>Hello World</H1>

<P>
Here is a new paragraph.

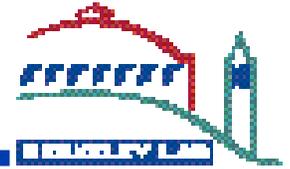
</BODY>
</HTML>
```

HTTP Daemon: know HTTP



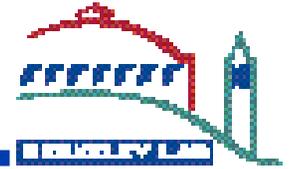
- **HyperText Transfer Protocol is the network protocol that transmits HTML (and other) documents**
- **RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1**
- **RFC 2617: HTTP Authentication: Basic and Digest Access Authentication**
- **RFC 2109: HTTP State Management (cookies)**
- **RFC 1945: HTTP/1.0 (informational)**

HTTP Messages



- **Communication consists of messages**
- **A message consists of a client request and a server response**
- **Request and response messages are “like email” (RFC 822) in that there is a header, a blank line, and the body.**

Client Requests

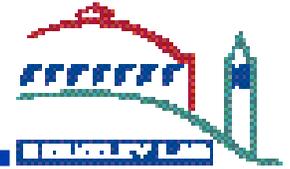


- ***A client establishes a connection to a server and sends a request message and waits for a response.***
- **Client Sends a Request-Line =
Method SP Request-URI SP HTTP-Version CRLF**
- **Along with Request Headers such as Host, User-Agent, Referer, etc.**
- **Methods = HEAD, GET, POST, etc.**

GET /pub/WWW/TheProject.html HTTP/1.1

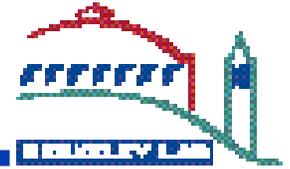
Host: www.w3.org

Client Header List



```
request-header = Accept                ; Section 14.1
                | Accept-Charset       ; Section 14.2
                | Accept-Encoding      ; Section 14.3
                | Accept-Language      ; Section 14.4
                | Authorization        ; Section 14.8
                | Expect               ; Section 14.20
                | From                 ; Section 14.22
                | Host                 ; Section 14.23
                | If-Match             ; Section 14.24
                | If-Modified-Since    ; Section 14.25
                | If-None-Match        ; Section 14.26
                | If-Range             ; Section 14.27
                | If-Unmodified-Since  ; Section 14.28
                | Max-Forwards         ; Section 14.31
                | Proxy-Authorization  ; Section 14.34
                | Range                ; Section 14.35
                | Referer              ; Section 14.36
                | TE                   ; Section 14.39
                | User-Agent           ; Section 14.43
```

Server Response

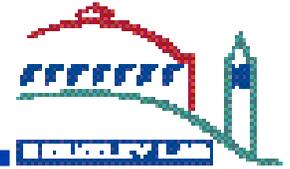


- **Server Sends a Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF**
- **Along with Response Headers and Entity Headers such as Location:, Server:, or Content-type:**

```
HTTP/1.1 200 OK
Date: Tue, 15 Jul 2002 02:51:51 GMT
Server: Apache/1.3.26 (Darwin) mod_macro/1.1.2
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html
```

```
<html><head><title>My Title</title></head>
<body>Wow!</body></html>
```

Server Status Codes



1xx: Informational - Request received, continuing process

2xx: Success - The action was successfully received, understood, and accepted

3xx: Redirection - Further action must be taken in order to complete the request

4xx: Client Error - The request contains bad syntax or cannot be fulfilled

5xx: Server Error - The server failed to fulfill an apparently valid request

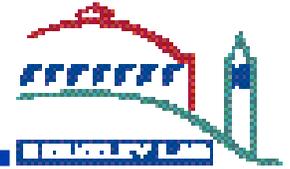
Most Common:

"200" ; Section 10.2.1: OK

"301" ; Section 10.3.2: Moved Permanently

"404" ; Section 10.4.5: Not Found

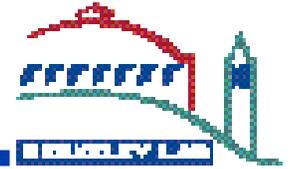
Statefulness



HTTP has no way of “remembering” anything about a previous connection.

Enter the “magic cookie”

HTTP Cookies



Content-type: text/html

**Set-Cookie: foo=bar; path=/; expires=Mon, 09-Dec-2003
13:46:00 GMT**

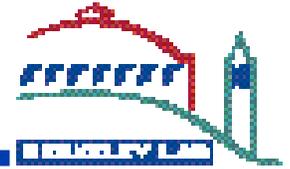
This header entry would result in a cookie named foo. The value of foo is bar. In addition, this cookie has a path of /, meaning that it is valid for the entire site, and it has an expiration date of Dec 9, 2003 at 1:46pm Greenwich Mean Time (or Universal Time). Provided the browser can understand this header, the cookie will be set.

When a cookie is sent from the browser to the server, the cookie header is changed slightly:

Content-type: text/html

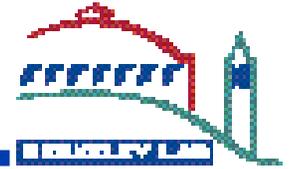
Cookie: foo=bar

Mostly Safe to Ignore for now



- **Ignore other “client programming”:** JavaScript, VBScript, JAVA Applets. We are striving for browser neutral (which is why we want to send plain HTML)
- **More complex HTML and relatives should be investigated for proper behavior in various browsers before counting on:** XML, XHTML, DHTML, CSS

HTML/HTTP Reference



HTML

- <http://www.w3c.org/MarkUp/Guide/>
- <http://www.w3.org/TR/1999/REC-html401-19991224/>

HTTP 1.1

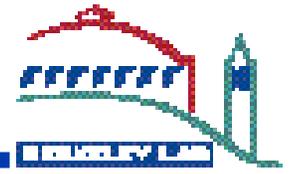
- <http://www.w3c.org/Protocols/rfc822/>
- <http://www.w3c.org/Protocols/rfc2616/rfc2616.html>

Basic/Digest Auth

- <ftp://ftp.isi.edu/in-notes/rfc2617.txt>

Cookies

- http://wp.netscape.com/newsref/std/cookie_spec.html
- <http://www.w3.org/Protocols/rfc2109/rfc2109.txt>
- <ftp://ftp.rfc-editor.org/in-notes/rfc2965.txt>

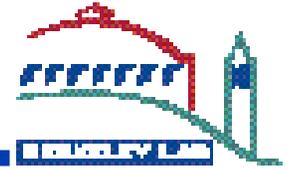




Web Server Programming

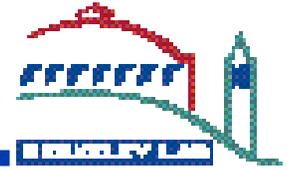
Towards Dynamic Content

Programming Options



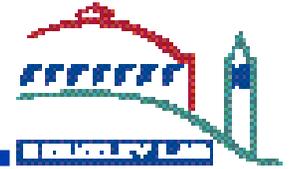
- **Server Side Includes (mod_include)**
- **CGI Programs (mod_cgi)**
- **Improved Page Parsing (PHP, ASP)**
- **Improved CGI (FastCGI, mod_perl Registry)**
- **Apache API via C & Perl**
- **C source code modification**

Server Side Includes Server Config



- **Options +Includes**
- **AddType text/html .shtml**
- **AddHandler server-parsed .shtml**

SSI Syntax



<!--#element attribute=value attribute=value ... -->

Sample Directives

#include (file|virtual)=value

<!--#include virtual="/footer.html" -->

#flastmod

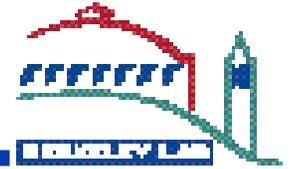
<!--#flastmod file="ssi.shtml" -->

#echo

<!--#echo var="LAST_MODIFIED" -->

#if, #elif, #else, #endif

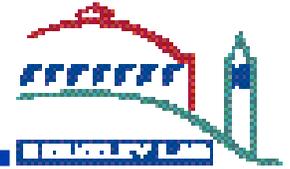
SSI Uses



- **Auto Header and Footers for standard look and feel**
- **Conditional loops based on variables such as browser type (maybe a special logo for IE users?)**
- **Last Modified timestamping**

#exec can run arbitrary programs, useful, but risky.

HTML Forms



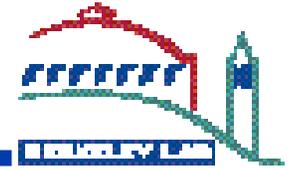
Key Elements

FORM

INPUT

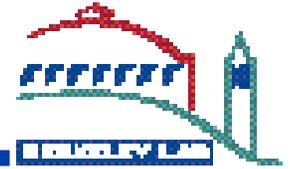
SELECT & OPTION

The Form Element



```
<!ELEMENT FORM - - ( %block; |SCRIPT)+ -(FORM) -- interactive form -->
<!ATTLIST FORM
  %attrs;                                -- %coreattrs, %i18n, %events --
  action      %URI;                       #REQUIRED -- server-side form handler --
  method      (GET|POST)                  GET       -- HTTP method used to submit the form--
  enctype     %ContentType; "application/x-www-form-urlencoded"
  accept      %ContentTypes; #IMPLIED -- list of MIME types for file upload --
  name        CDATA                       #IMPLIED -- name of form for scripting --
  onsubmit    %Script;                    #IMPLIED -- the form was submitted --
  onreset     %Script;                    #IMPLIED -- the form was reset --
  accept-charset %Charsets; #IMPLIED -- list of supported charsets --
>
```

A Basic Form



```
<HTML>
<HEAD>
<TITLE>Form 1 - Basic</TITLE>
</HEAD>

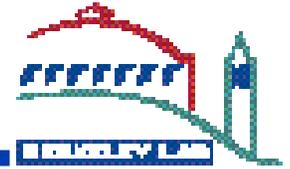
<H1>Form 1- Basic</H1>

<BODY>

<FORM action="http://cisdemo.lbl.gov/cgi-bin/get_name.sh" method="get">
  <P>
    First name: <INPUT type="text" name="firstname"><BR>
    Last name: <INPUT type="text" name="lastname"><BR>
    <INPUT type="radio" name="sex" value="Male"> Male<BR>
    <INPUT type="radio" name="sex" value="Female"> Female<BR>
    <INPUT type="submit" value="Send"> <INPUT type="reset">
  </P>
</FORM>

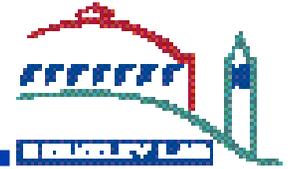
</BODY>
</HTML>
```

The Common Gateway Interface



- **The Common Gateway Interface (CGI) is a standard for communication between a web server and an external executable program.**
- **The program can be in any language which is executable by server host; script or binary**
- **The server communicates with the CGI by setting environment variables and possibly sending data to the program STDIN**
- **The STDOUT of the program is sent to the browser by the server. This is usually HTML but could be any valid MIME type: text, a graphic, etc.**

CGI: server configuration



- **Usually just:**

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

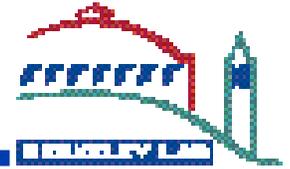
- **But could be done:**

```
<Directory /usr/local/apache/htdocs/somedir>  
    Options +ExecCGI  
</Directory>
```

- **Or:**

```
AddHandler cgi-script cgi pl
```

CGI Hello World

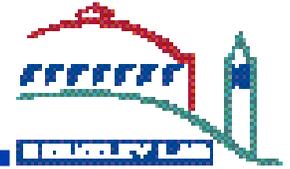


```
#!/bin/sh
```

```
echo "Content-type: text/plain
```

```
Hello World"
```

CGI Environment Variables



The CGI standard has a bunch of useful environment variables that will be defined when your script is called.

Important ones:

REMOTE_ADDR

Getting Info:

QUERY_STRING

PATH_INFO

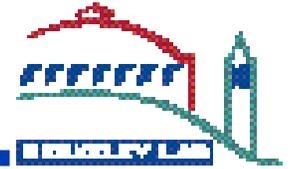
CGI behind the scenes



The listening httpd process forks a copy of itself then execs the CGI program which runs in it's place, connected to the server by STDIN/STDOUT.

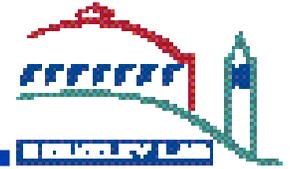
- **This is simple and straight forward**
- **Info is shared by ENV variables and STDIN/STDOUT**
- **No need to worry about “leaky memory”**
- **“slow”**

CGI Cookies Revisited



CGIs can send HTTP header information, so they are perfect for manipulating cookies.

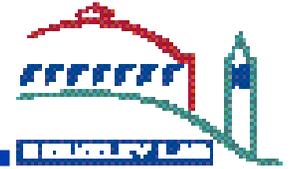
Web Sessions



You've seen Basic authentication.

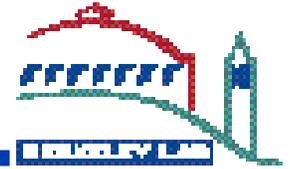
Web sessions like Yahoo maintain your login and logout status via cookies.

CGI: FORMS, GET/POST and parameters



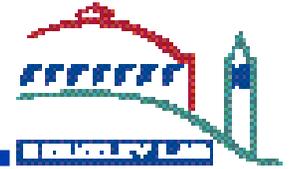
**Two main ways to get data to a CGI from a form,
GET and POST**

get_name.html



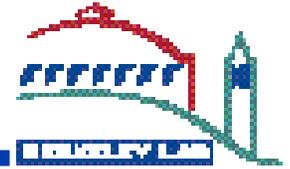
The beginnings of a guest book, let's start with a shell script, then move to something a bit more complex.

get_name Analysis



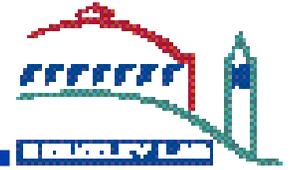
- If we wrote this to a file, we would have a guestbook.
- Clearly, it is going to get tedious parsing `QUERY_STRING` again and again
- What if they had used the `POST` method? We wouldn't see the data because we are only looking for `GETs`
- There are many `www` libraries already written in almost every language, let's take advantage of one of them.
- **PROBLEM:** mixing `HTML` and code looks messy

Using perl's CGI.pm



- **Many built in functions and short-cuts (though the short cuts do make it a bit confusing for new users).**
- **Handles all parameters automatically whether GET or POST**
- **Will default to POSTing to itself, with interesting results**

What Language is best for CGI?

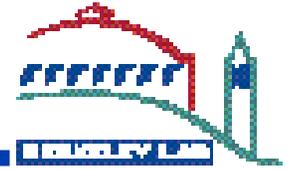


C doesn't have to be interpreted (costs time).

There are lots of libraries for perl, but there are a bunch for other languages, tcl, python, etc.

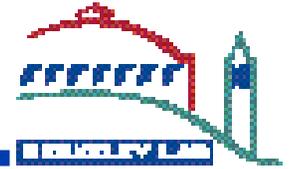
JAVA is probably not a good CGI language because of slow launch times.

Writing Secure CGIs



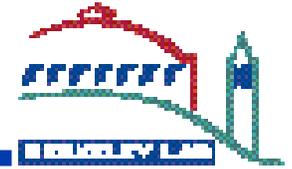
- **Watch all eval and system calls which incorporate user data**
- **Don't trust any data sent by the user**
- **In perl, beware of instances where the shell might be called (system, back ticks, open)**
- **su-Exec helps in multiuser content owner situations**
- **SSL helps keep transactions private, but doesn't prevent clients from abusing you**

CGI Exploits



- **Crashing pages**
- **Server crashes**
- **Running Arbitrary Code**
- **“forking an xterm”**

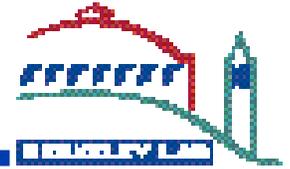
Beyond CGI



Today, most apps are written inside server interpreters now, such as PHP, mod_perl, or various application servers such as tomcat for JSP and servlets.

- **Memory resident state (same program is still running)**
- **Faster (no fork/exec) and often dynamic pages are cached as well**
- **Can access other server internal functions (like authentication and URL rewriting)**

ASP/ADP/JSP

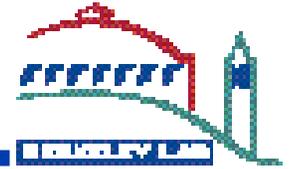


SYNTAX

<% ...code here... %>

<%= ... code with result printed %>

Define own tags <yourtag ...>



SYNTAX

[+ Perl code with output +]

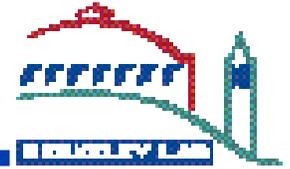
[- Perl code don't print output -]

[! Perl code executed only first time !]

[# Some Text #] (Comments)

[\$ Cmd Arg \$] (Meta-Commands: special if, elsif, else, endif, etc...)

Web Programming Reference



Security: <http://www.w3.org/Security/Faq/www-security-faq.html>

- Apache: <http://httpd.apache.org>
- CGI: <http://hoofoo.ncsa.uiuc.edu/cgi/intro.html>
- Mod-perl: <http://perl.apache.org>
- PHP: <http://www.php.net>
<http://www.php.net/manual/en/>
- Embperl: <http://perl.apache.org/embperl>
- ASP: <http://www.apache-asp.org/>